EE554/CSE586 Spring 2018 **Course Project** Due Date: Sun Apr 29, 11:59PM

Transfer Learning with Convolution Neural Networks

The goal of this project is to gain practical experience with convolutional neural networks. The task is to do transfer learning to take an existing pre-trained classifier and fine tune it to handle a different dataset than it was originally trained on. This project is motivated in part by the paper https://arxiv.org/abs/1403.6382 "CNN Features off-the-shelf: an Astounding Baseline for Recognition" and by the course notes at http://cs231n.github.io/transfer-learning/ .

Nearly no-one writes CNN software from scratch any more, although I hope you feel confident that using what we learned in class you **could** write and train a simple CNN composed of computational modules we have discussed. It would run slowly however, unless you spent a lot of effort vectorizing and optimizing the code for efficiency. Luckily, existing CNN libraries have done all that optimization work already!

Furthermore, not many people train a CNN from scratch either. The computational resources required to train a (large) network with millions of parameters, using millions of labeled training images, makes it prohibitively expensive for rapid prototyping of a new classifier on a new dataset. Luckily (again!), other groups with more resources and more time on their hands have already trained classifiers on large datasets like ImageNet, and these pre-trained classifiers are available to read in and use.

In this project, you are tasked with trying out CNN transfer learning. There is a lot of freedom to choose the CNN software, the pre-trained CNN, the dataset, and so on. It is hoped that with the amount of variation possible in the space of choices you can make, each group has the freedom to be choose something interesting to them, and that is likely to be something different from other groups. As part of the project "deliverable", you will write a report describing what you did and what choices you made, how it worked and so on.

The rough steps of the project:

1. Choose a framework for implementing CNNs.

Recent versions of Matlab have CNN software built in. The page https://www.mathworks.com/help/nnet/convolutional-neural-networks.html has links to resources and tutorials that describe how to use it. In particular, I notice under Topics that there is a set of tutorials on how to "Work With Pretrained Networks", which sounds pretty relevant.

Last year my class used MatConvNet from the Vlfeat group: http://www.vlfeat.org/matconvnet/ . This is a library of code and examples that you can call from Matlab (it was developed before mathworks added CNN functionality directly). It's a little yucky, in that you have to compile some parts of the code for your platform, unless you can find a precompiled binary for your machine. MatConvNet provides the ability to design and train your own CNNs, but, more importantly for this project, also has a library of pre-trained models http://www.vlfeat.org/matconvnet/pretrained/ .

If you prefer to use an "industrial strength" CNN toolkit, you could consider installing Caffe or Tensorflow (or Torch or Theano or …). I have never used any of them, and I understand there is a steep learning curve for most if not all of them. However, it you intend to do research in the future using deep learning / convnets, now might be the time to dive in.

2. Choose a pre-trained CNN

This will be the CNN that provides the features you will be using for classification. By using an already-competent classifier and "freezing" most of the parameters, you avoid the need for large amounts of training data and time. The library you choose for implementing CNNs will no doubt have several to choose from. For example, see the pages at:

MatConvNet: http://www.vlfeat.org/matconvnet/pretrained/
Matlab: https://www.mathworks.com/help/nnet/ug/pretrained-convolutional-neural-networks.html

There will also typically be ways to read in other models that have been stored in a common format, for example importing from Caffe. This opens up the ability to access a constantly growing list of user-posted models known as Caffe Model Zoo https://github.com/BVLC/caffe/wiki/Model-Zoo .

3. Choose a dataset to work with

This is supervised learning, so we need a dataset that has labeled images. There are many available, ranging from two classes (cats vs dogs) to many classes (Google's OpenImage dataset has 6000 classes). The only constraint I place on what to choose is that you have to use a different dataset than the one that your pre-trained CNN was trained with.

Some ideas for datasets:

http://deeplearning.net/datasets/
http://www.cs.utexas.edu/~grauman/courses/spring2008/datasets.htm
http://www.robots.ox.ac.uk/~vgg/data/

Any of the datasets used in the "CNN Features Off-the-shelf" paper ; in fact you perhaps could try to replicate one of their experiments.

Datasets from Kaggle competitions. For example:
   https://www.kaggle.com/c/cifar-10
   https://www.kaggle.com/c/dogs-vs-cats/data

Of course you are also welcome to generate your own dataset. One year I had my class learn to recognize Chinese characters for numbers 1-10 , part of which involved generating their own dataset of characters in different font styles downloaded from https://www.freechinesefont.com/ .

Yet another dataset idea is the prealigned and cropped face (head) data used by James Hay's GAtech class to learn face detection https://www.cc.gatech.edu/~hays/compvision/proj5/. That project was not about deep learning, but the dataset of face images looks rather nicely preprocessed. He also includes a negative dataset of images that do not contain faces, from which huge numbers of negative examples could be cropped at random.

One thing to keep in mind is that the more similar the dataset you will use is to the dataset that was used to train the pre-trained classifier, the better the results will likely be. An object recognition classifier trained on natural images in ImageNet will not necessarily transfer well to medical images or astronomy pictures. Another thing to keep in mind is that you might need to manipulate images from the new dataset into the format expected by the pre-trained classifier. For example, if the pre-training CNN expects images of size 100x100 to come in, and the images in your new dataset are a different size, then you need to turn them into 100x100 by resizing or cropping.

4. Fine tune your network

There are choices to make at this stage too! Do you want to use the pre-trained network to just extract feature vectors that you then use as input to another classifier, or do you want to actually fine tune parameters of the network, and if so, how many layers do you want to fine tune? (see Figure 1).

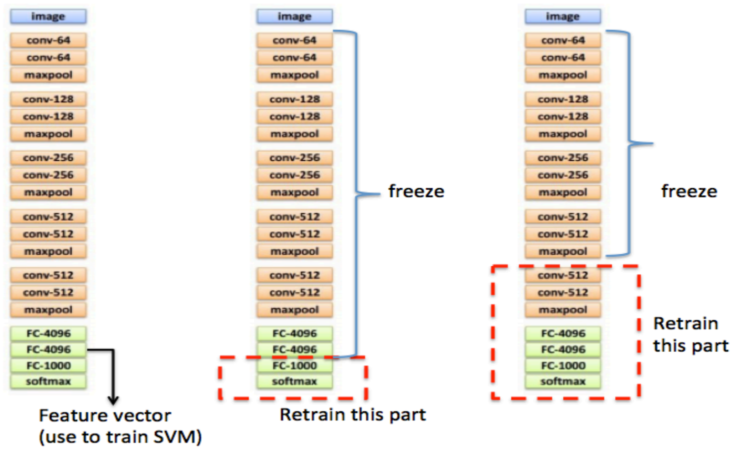

Figure 1: Three examples of how you might transform a pre-trained CNN into a classifier for another dataset.

The simplest thing is to run training images in the forward direction through the pre-trained CNN, and take the output values of some layer(s) near the end as a vector "code" representing the image. Since you also have the correct class labels for those training image, you can then send them as labeled training data to train, for example, a support vector machine. This would be an easy way of trying to build a classifier for two-class problems. By the way, if you want to do two-class classification, you don't have to choose datasets with only two classes in them. Any multi-class dataset can be turned into a set of two-class problems; for example, just taking two of the classes and learning class A vs class B, or choosing just one class you care about and learning to distinguish class A from all of the other classes.

Perhaps a more natural approach that stays within the context of deep learning is to replace the final fully connect and softmax layer of the network with your own layers, that you then train using backpropagation. Note that the rest of the parameters in the network should be "frozen" and thus not subject to change during backprop (you could set their learning rate to 0, or you could even cache the feature vector code mentioned in the paragraph above and use it as input to learn a single small fully-connect convnet). The CNN architecture shown in the picture was trained on ImageNet, so the last fully connect layer produces 1000 numbers, one for each of the 1000 ImageNet classes. If your dataset has a different number of classes, then you would replace this with a fully connect layer that outputs the number of classes you are learning. As discussed in class lectures, training a final fully connect layer is equivalent to learning a multi-class linear classifier, in this case using the 4096-dimensional feature vector space produced by the pre-training CNN.

Finally, if you are ambitious, you could also try "unfreezing" additional layers near the end of the network and fine tuning their parameters as well, using backpropagation. Since you are increasing the number of parameters you need to learn, you would expect to need to use a moderate-to-large training set to learn them well.

5. Evaluation

So, how well did you do? In addition to training data, many labeled datasets also include labeled testing data that you can use to evaluate performance of the fine-tuned classifier. If only one big pool of training data is given, then it is common to split it yourself into one subset for training and to reserve the remaining subset for testing/evaluation… maybe a 60-40 split between training and testing. In any case, you will want to come up with some sort of quantitative evaluation measures that determine how well your new classifier is doing. Some common choices are classification accuracy, top-k classification accuracy (number of times the correct class is in the top k highest scores coming out of the classifier divided by the number of tests), and confusion matrix.

What to turn in?

1) Submit your code in a zip file, on Canvas. Just the code please… don't upload the dataset you used, that will take up tons of space on our course site.

2) Include in the zip file a pdf file containing the project report. Some things I'd like to see in the report:

a) Names of group members (important!)

b) A short paragraph summarizing in your own words what the project was about, what tasks you performed, what results you expected to achieve, and how it was related to our course material.

c) Explain each of the design choices you made for steps 1 through 4 above. That is, what software did you use, what was the pretrained classifier, where did you get your data (this would be a good place to show some example images, and to include a link to where the dataset comes from if you got it from the web), how did you do the transfer learning? Pay particular attention to explaining things you had to do to make things work that might not be obvious at first. For example, did you have to do any normalization or data augmentation to the input data or did you just send in the raw images? Did you do you own training-testing split or does the dataset come that way already? If you used backpropagation to fine-tune some of layers of the CNN, what layers were they, what learning rate did you use, and how did you check that it was doing something useful (e.g. loss decreasing across epochs)?

d) Show your quantitative evaluation results and discuss. High-level question: did it work? Did the results come out as well as you expected? If it is a multi-class problem, does the classifier do better with some classes than others? What are the main confusions between classes, and is that because they look like each other (show us), or is there no apparent reason why? Are there any lessons learned that would cause you to do something differently next time?

e) A brief statement about what each group member contributed to the project. Division of labor means you can divide up tasks, but try to ensure that everyone contributed materially to the work and had a fair chance to learn something from the exercise.